Comparison of Multi-Server Queueing System Models

Md Farhan Ishmam[1] and Md Toshaduc Rahman[2]

[1]Department of Computer Engineering, Islamic University of Technology, Gazipur, Bangladesh
farhanishmam@iut-dhaka.edu

[2]Department of Computer Engineering, Islamic University of Technology, Gazipur, Bangladesh
toshaduc@iut-dhaka.edu

ABSTRACT

The multi-server queuing system functions similarly to its single-server counterparts but has crucial design choices. In this paper, we shall have a look at those design choices in constructing a multi-server queuing system. The most important element of a queueing system i.e., the queue can be subject to such design choices. This paper looks at two scenarios of such systems, with a single queue and with two separate queues. A comparison shall be drawn, with a verdict on the better system.

KEYWORDS

Single-Server Queueing System (SSQS), Multi-Server Queueing System, Simulation (MSQS), Arrival Event, Departure Event, Exponential Distribution, Queueing Delay, System Delay, Jockeying, Switch

1. Introduction

The multi-server queueing system consists of more than one server. The servers may or may not be interconnected. Such a system can have as few as two independent servers (as in this paper) or as many as hundreds of servers interconnected in a complex design (example - Job Shop Model). The huge complexity behind designing and implementing a multi-server queueing system has given us opportunities to make some design choices in order to make the server more efficient i.e. reduce system delay.

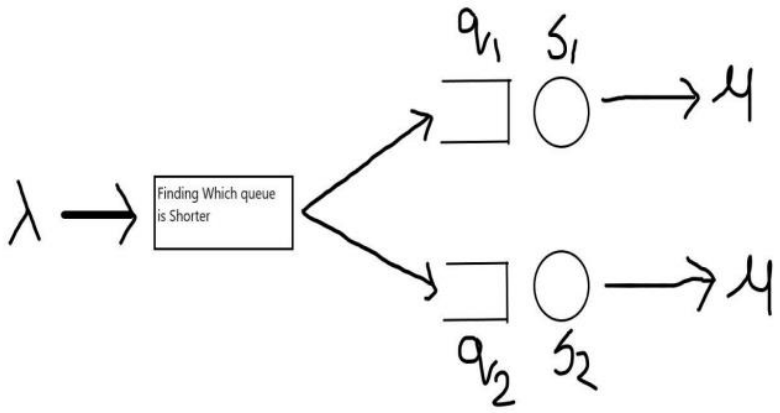
The queue is by far the most important element of your system. The queueing procedure can vary by changing the working mechanism of the queue i.e., changing it from standard FIFO queue to Round-Robin or so-on. Another factor we should consider is the number of queues in our system. Usually, we assume that each server will have its own independent queue. However, we can also make a system with a shared queue. Usually, such a shared queue is seen at the item arrival part. We shall compare such a system with a shared queue to another system with independent queues. For the sake of simplicity, we will be dealing with two servers only.

We will first look at a detailed overview of the two systems. Then a description of our simulation program is given by presenting the implementational details and describing the classes of the system. Then we will go through the output statistics of our model and analyze the external factors that can affect our results. We will look at the dataset obtained from running the simulation programs multiple times, and visualize them by graphs. Finally, the verdict on the better queueing model will be given.

2. System Description

The systems follow standard queueing systems with two servers. We will have two scenarios - the first one has two servers with their separate queues and the second one has two servers with a shared queue.

Scenario - 1:



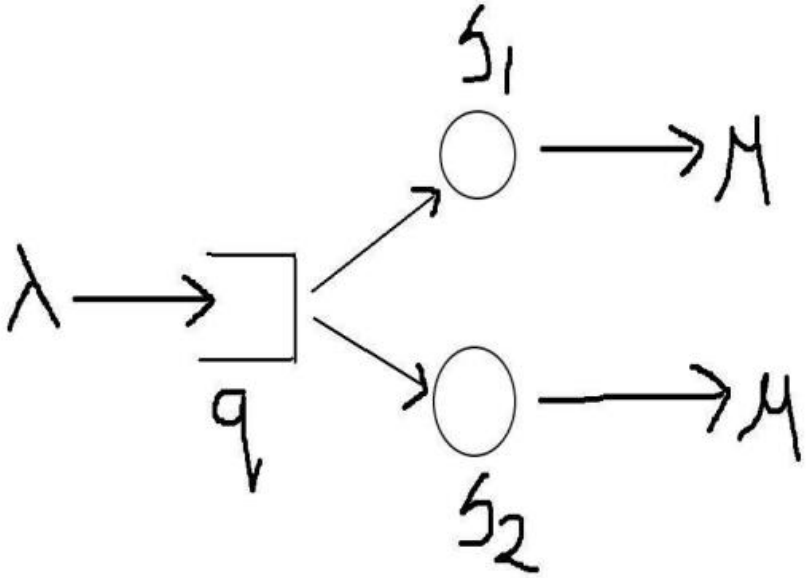
In the first scenario, there is a single arrival rate for the two servers. Upon arrival, the shortest queue has to be found and the item is enqueued or given service at that queue. The departure rates are different for the servers but for the sake of simplicity, we use equal departure rates. The traffic intensity is measured by dividing the arrival rate by one of those departure rates. The system differs from the single server queueing system by serving the items parallelly at the same time.

It is to be noted that this two-server model is more efficient than two Single Server Queuing Systems. In this model, a decision is made in picking the shortest queue which prevents items from unnecessarily waiting at longer queues and thus having less queueing delay. Lesser queueing delay leads to lesser system delay. So, to make a valid comparison between the number of queues of two multi-server queueing systems, the decision of finding the shortest queue is included in this scenario.

Another factor to be considered is the type of queue. For this scenario, we use FIFO or the First In First Out queue. Other forms of queue such as circular queue, priority queue, or Round Robin queue can be implemented, and would vastly impact the system delay. However, this comparison is done with FIFO queues in particular as they are the most common form of a queue. Scenario-2 will also have a FIFO queue.

Finally, jockeying is not allowed in this system. Jockeying means changing queues after being enqueued to a particular queue. Jockeying is done to reduce queueing delay. This feature has also been excluded for the sake of simplicity and also to put more emphasis on the number of queues to reduce queueing delay than other complementary factors.

Scenario - 2:



In the second scenario, there is a single queue with a particular arrival rate. This single queue is shared by the two servers. The queue will check the status of the servers and provide service to the idle server. The queue will also be a FIFO queue. Here, there is a single arrival rate for the queue but the departure rates may vary for each server. For the sake of simplicity, we take equal departure rates for the servers. The traffic intensity is calculated by dividing the arrival rate by one of those departure rates. Similar to scenario-1 items can receive service simultaneously.

Theoretically, scenario-2 should be the more efficient system as in scenario-1 there might be a case that there are items enqueued in the first queue while the second server is idle. Scenario-2 avoids such a problem by sharing the queue. Using the simulation program, we compare the waiting time and delays for each scenario.

3. Simulation Program Description

The program will consist of four major classes - **the event class, the server class, the scheduler class, and the queue class**. For scenario-1, there is another class called **service facility** which encapsulates all the servers of the system.

The simulation program for both the scenarios are modifications of the simulation program of the single server queueing system. The multi-server queueing models can be treated as an extension of the single-server program. The core functions of handling the arrival, departure events, initializing servers, calculating output statistics and generating reports are kept the same. The event, scheduler, and queue classes remain unchanged.

For scenario-1, there is a switch which takes the arrival and enqueues to the server with the shortest queue. A function called arrivalSwitch() has been added to server class to perform this task. As seen in the diagram, two servers are needed for the program. The arrivalSwitch() function also has to know the servers of the system. Hence, each server must keep a pointer to the other server. The arrival server checks the queue length of both the servers and then determines the shortest server.

A composition super-class of the two servers was created for convenience. The service facility class is composed of these two servers and is responsible for initializing the servers and generating the reports. The service facility class has extended functionality as it can be used to add more servers to the system and can hence be used to construct complex designs.

The program creates a trace file and a report file as out. The reports are generated separately for each server. It is to be noted that the final report data is the average or summation of the reports from the two servers.

For scenario-2, two approaches can be followed. The first approach is the same as scenario-1 but the queue is declared as a static object. Hence, the queue objects' memory is shared among all the instances of the queue class. However, in this program, a straightforward approach of adding an extra server status in the server class was done. In the server class, there is an instance of the queue. So, by default, each server has its own independent queue. The server class is modified by adding an extra server status variable for the second server. In this way the same queue is shared by the two servers. However, while providing service, both servers need to be checked, and then items are enqueued to the idle server.

The reports are generated together. A little modification is done with the report generation code, so that the format has both server statuses in the trace file and the output statistics variables consider the status of both the servers.

4. Output Statistics

We will introduce some new terms and symbols for the output statistics that we will calculate:

λ = the arrival rate of the customers
μ = the departure rate of the customers
$\rho = \lambda/\mu$ = traffic intensity which is the ratio of arrival and departure rate
L_q = time-averaged number of customers in the queue
L_u = time-averaged server utilization
L_s = time-averaged number of customers in the system
W_q = job-averaged queueing delay
W_s = job-averaged system delay

The traffic intensity is treated as the input to the system, while the five output statistics variables are our required outputs. A trace file is generated for the servers in both the scenarios. The trace file contains the arrival, service and departure time of all items processed in the server. The reader can verify the values generated in the report by parsing the trace file, and performing the required mathematical operations to generate the given output statistics. However, it is not recommended as we are confident in our procedure of generating the report.

5. Hyperparameters and External Factors

We will now have a look at the other parameters that can impact the performance of our model. The explanation of these factors is given below:

i) Number of Items: The number of items tends to show no change in the output statistics of the model at lower traffic intensity. However, at traffic intensity higher than 0.5 there is a significant impact. As traffic intensity increases, the impact gets higher.

If the number of items processed by the program is too small, then the time taken to run the program will be less but the output statistics vary a lot due to randomness. Again, if the number of items is too large, then the time taken will be higher, but the program will produce more consistent results. The impact is significantly noticeable at high traffic intensity such as 0.99 when delays, and queue size should theoretically get close to infinity. Practically, the output values tend to reach a high value but as the number of items processed is limited, the output

value stays within a finite limit. Access to more computational resources, can enable us to process a higher number of items and thus produce more accurate results.

The program will simulate the queueing process of 1000 items. This hyperparameter was tuned by running the model multiple times and finding the best value for optimal results. However, values like 2000, 3000 and so on can be considered but don't change the output statistics significantly.

ii) **Output Statistics Metrics:** While calculating the traffic intensity, only one of the departure rates were considered. The departure rates were also considered to be equal. These design choices were done for simplicity but can be changed to create more consistent results. For example - instead of calculating the traffic intensity using a single departure rate, the summation of the departure rates can be taken.
For scenario-1, there are two sets of output statistics; one for each server. For all the variables except server utilization the values of server-1 and server-2 are added. Only, for server utilization, the values are averaged. The results were shown to be consistent if a single report was created by calculating the output statistics combinedly for both servers.

iii) **Multiple Runs of the Program:** The program was tested multiple times and produced consistent results. Due to the randomness, there can be slight variations in the output statistics produced in each run. However, by processing a high number of items, the randomness was reduced to an insignificant margin. The results are almost identical to the first four decimal places in each iteration and have less than 0.01% error which can be neglected. Hence, the results were calculated by running the program for a particular value of traffic intensity just once and multiple runs for the same input values was not required.

iv) **Arrival and Departure Rates:** Traffic intensity was calculated by changing the arrival rate and keeping the departure rate constant. However, this design choice can impact some values such as number of customers in the queue. The traffic intensity can also be changed by varying the departure rate, or by changing both.
The values of arrival and departure means can also impact the out statistics. Using a different scale can change the results. For example – using 1.0 and 10.0 as arrival and departure mean has the same traffic intensity value as using 10.0 and 100.0. However, the output statistics will produce different results. It is to be noted that even by changing the scale of the arrival and departure rates, the produced graphs of the two scenarios stays the same. For the sake of simplicity, we chose 10.0 as our fixed departure rate and varied arrival rate from 1.0 to 9.9 for both of the scenarios.

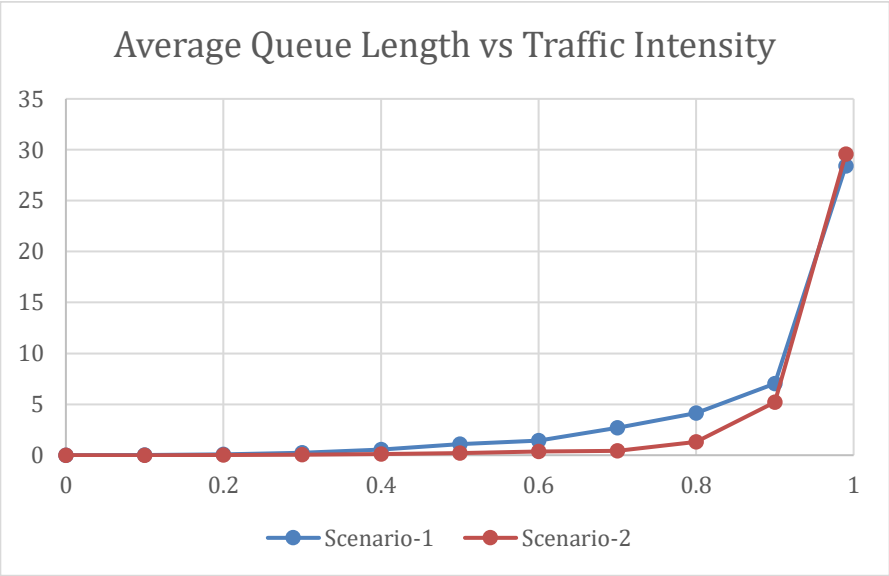
6. Simulation Results

The simulation results were taken by running the simulation 11 times for different values of traffic intensity ρ. As mentioned earlier, the departure rate was fixed to 10.0 and by changing the arrival rate, we got different values of traffic intensity. The traffic intensity ranges from 0 to 0.99 with a 0.1 increase in each step, except the last step with a 0.09 increase. The same values of traffic intensity were used for both the scenarios. The results of the simulation program is given below:
:

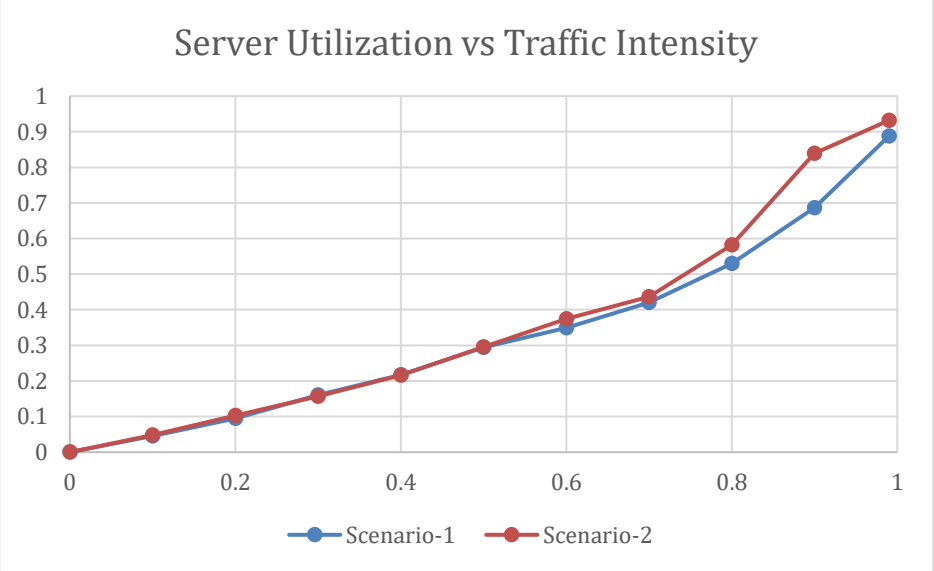
Name	Symbol, Eq	1	2	3	4	5	6	7	8	9	10	11
Arrival Rate	λ	0	1	2	3	4	5	6	7	8	9	9.9
Server-1 Departure Rate	$\mu 1$	10	10	10	10	10	10	10	10	10	10	10
Server-2 Departure Rate	$\mu 2$	10	10	10	10	10	10	10	10	10	10	10
Server-1 Traffic Intensity	$\rho 1 = \lambda/\mu 1$	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99
Server-2 Traffic Intensity	$\rho 2 = \lambda/\mu 2$	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99
Traffic Intensity	$\rho = (\rho 1 + \rho 2)/2$	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99
Scenario-1 Simulation Results												
	Average Server Output Statistics											
Average Queue Length	$L_q = L_{q1} + L_{q2}$	0	0.0174	0.0849	0.2541	0.5577	1.0829	1.4467	2.6846	4.1344	7.0159	28.3997
Average Server Utilization	$L_u = (L_{u1}+L_{u2})/2$	0	0.0455	0.0951	0.1603	0.2174	0.2940	0.3493	0.4200	0.5295	0.6860	0.8880
Average Number of Items	$L_s = L_{s1} + L_{s2}$	0	0.2252	0.4999	0.9556	1.5487	2.4453	2.9528	4.6615	6.4610	######	50.8073
Average Queueing Delay	$W_q = W_{q1} + W_{q2}$	0	0.0173	0.0417	0.0816	0.1346	0.2028	0.2374	0.3731	0.5150	0.7816	3.0892
Average System Delay	$W_s = W_{s1} + W_{s2}$	0	0.1972	0.2367	0.2674	0.3198	0.3898	0.4152	0.5574	0.6923	0.9675	3.3124
Scenario-2 Simulation Results												
	Server Output Statistics											
Average Queue Length	L_q	0	0.0005	0.0082	0.03661	0.09804	0.219	0.3511	0.4273	1.2943	5.2218	29.5696
Average Server Utilization	L_u	0	0.0482	0.1028	0.15713	0.21579	0.295	0.3744	0.4368	0.582	0.8395	0.93197
Average Number of Items	L_s	0	0.0973	0.222	0.38747	0.62767	1.027	1.451	1.6683	3.7526	12.123	61.0031
Average Queueing Delay	W_q	0	0.0005	0.0048	0.01514	0.03138	0.056	0.0751	0.0814	0.1967	0.6208	3.20523
Average System Delay	W_s	0	0.0923	0.0919	0.0995	0.11373	0.139	0.1602	0.1629	0.2873	0.7205	3.30626

7. Graphical Comparison of Scenario-1 and Scenario-2

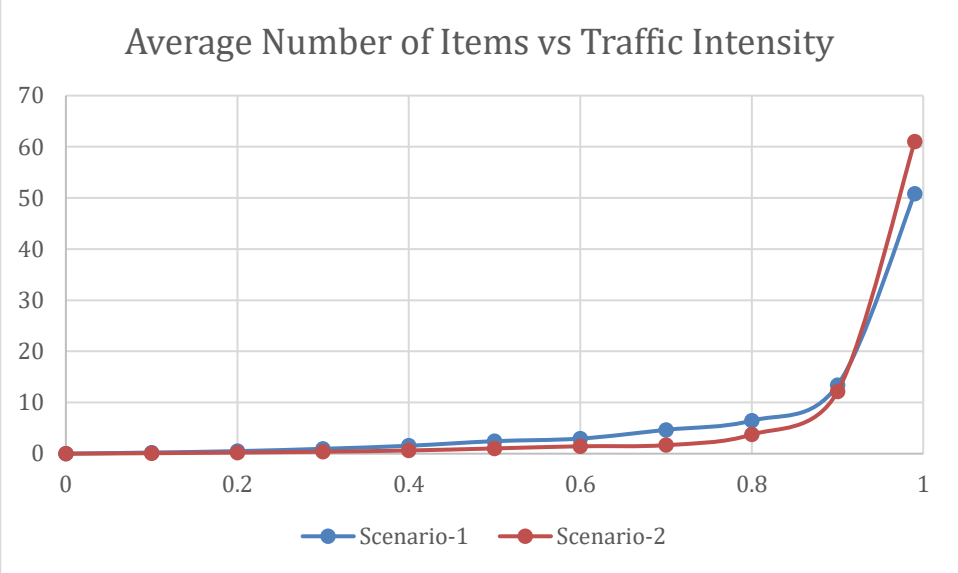
Time Averaged Queue Length: The simulation results for both the scenarios were similar up to 0.4 traffic intensity. From 0.5 traffic intensity Scenario-1 has a higher queue length. The difference kept increasing as the traffic intensity reached higher. At 0.99 traffic intensity both scenarios had similar queue length as the items processed were finite in number. The produced results are consistent to our theoretical findings as it was predicted that scenario-1 would have a higher queue length because of the separate queues leading to inefficiency of the system.



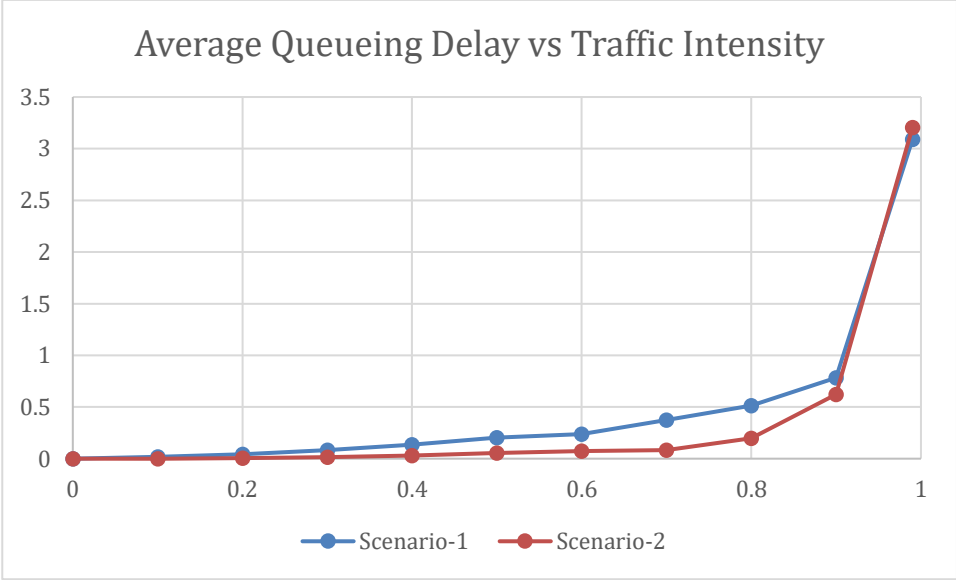
Time Averaged Server Utilization: The server utilization of the scenarios was similar up to 0.7 traffic intensity as both systems used their servers linearly. However, the server utilization of scenario-2 bumped higher at 0.8 and even higher at 0.9. This is due to the fact of using a shared queue keeps the server busy more often. Again, the values were seen to be similar at 0.99 traffic intensity as finite number of items were processed by the simulation system.



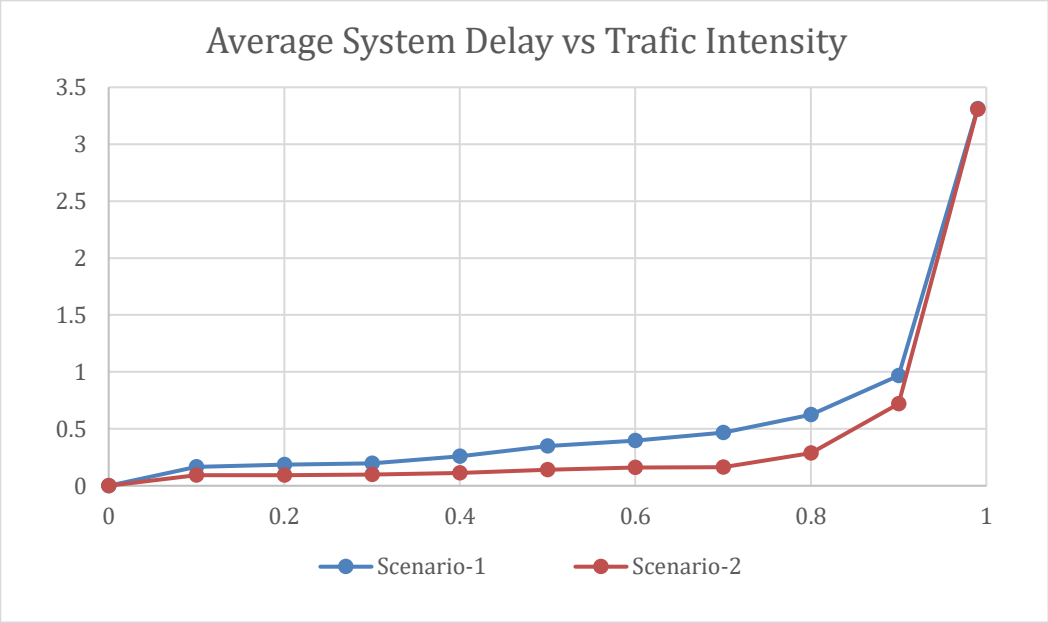
Time Averaged No of Items in the System: Similar to the comparison of queue length, the number of items in the system was similar up to 0.4 traffic intensity. But the number of items in scenario-1 started being higher at latter values of traffic intensity. Just like previous cases, at 0.99 traffic intensity, the values got closer. However, scenario-2 was significantly higher than scenario-1 in this case. The behavior of the system varies due to implementational details at really high values of traffic intensity as finite number of items were processed.



Job Averaged Queueing Delay: The comparison of queueing delay is similar to the previous cases. However, in this case, a difference is seen from 0.3 traffic intensity. This can further confirm that the inefficiency of scenario-1 is due to having higher queueing delay.



Job Averaged System Delay: The average system delay is identical to average queueing delay. Scenario-1 has a significantly higher system delay than Scenario-2. The differences are identical to the differences seen in queueing delay.



9. Conclusion:

Scenario-1 has higher queueing delay, system delay, queue length, number of items in the server and has lower server utilization than scenario-2. Due to this reason, we can safely conclude that scenario-2 is the better system. It was almost observed that the systems perform similarly at lower values of traffic intensity. However, the difference is noticeable as the value of traffic intensity increases. Another key observation is the converging values at 0.99 and higher traffic intensity. Theoretically as traffic intensity gets close to 1, those output statistics reach close to infinity. However, the report doesn't produce a huge number as we are performing the simulation for a finite number of items. Increasing the number of items can provide us better results at really high values of traffic intensity. Overall, the program successfully simulated the given scenarios.

REFERENCES:

[1] Simulation Modeling and Analysis by Averill Law, 4th Edition

[2] Discrete Event Simulation: A First Course by Lawrence Leemis, December 2004 Revision

[3] Project Link: https://github.com/Aplycaebous/Multi-Server-Queueing-System-Model-Comparism